# IUT de Paris Descartes – Base de la Programmation Objet

Travaux Dirigés – Sujet n°9

## Objectifs

— Définir une hiérarchie de classes
— Exploiter le sous-typage

## Un simple tableur

Il s'agit de réaliser une classe permettant la gestion d'un tableur à la "excel" n'offrant qu'un nombre très limité de fonctionnalités.

Les cellules composant un tableur pourront être vides, contenir une valeur entière constante ou contenir la somme de deux autres cellules. Pour limiter les cas d'erreur de ce dernier cas, les cellules vides seront évaluées par défaut à zéro et ainsi, elles pourront intervenir dans une somme. Les cellules du tableur seront désignées par leur numéro de ligne et de colonne.

Enfin, un tableur sera caractérisé par son nombre de lignes et de colonnes et ses cellules seront initialement toutes vides. Les seules fonctionnalités requises pour le tableur sont le positionnement d'une cellule (la rendant soit vide, soit constante, soit égale à une somme) et son affichage. Une cellule vide sera représentée par le caractère '`.`' et les autres cellules par la valeur entière qui leur correspond.

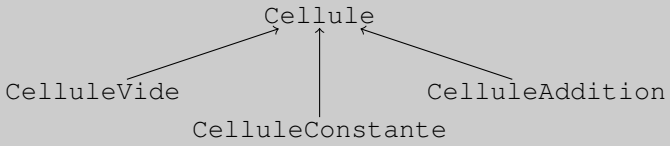
1. Définissez le prototype des méthodes publiques de la classe `Tableur` requises pour la spécification ci-dessus. Vous pourrez vous servir de la classe `Coord` permettant la représentation de coordonnées (le code est fournie en annexe et le fichier correspondant est sur le serveur commun). Cette classe (ainsi que toutes celles que vous créerez lors de cette séance) est définie dans la paquetage `tableur`.

   **Solution:**

   ```
   // Tableur.java
   package tableur;

   public class Tableur {
     public Tableur(int nbLignes, int nbColonnes) {
     }

     public void set(Coord c, Cellule cell) {
     }

     public String toString() {
     }
   }
   ```

2. Définissez la hiérarchie de classes nécessaire pour représenter les différents types de cellules ainsi que les services minimaux que chacune doit proposer. Écrivez la classe étant à la racine de cette hiérarchie.

**Solution:** La hiérarchie doit ressembler à cela :




Les services minimaux que doit offrir toutes les classes sont :

```
// Cellule.java
package tableur;

public abstract class Cellule {
  public abstract int évaluer();

  public String toString() {
    return Integer.toString(évaluer());
  }
}
```

Le code de la méthode d'évaluation ne peut pas être défini ici. En conséquence, la méthode (ainsi que la classe) est déclarée abstraite. La méthode `toString` est spécialisée ici pour prévoir un traitement par défaut. Si celui-ci ne convient pas, les sous-classes pourront la spécialiser comme elles le souhaitent.

3. Définissez et codez la sous-classe représentant les cellules vides.

**Solution:**

```
// CelluleVide.java
package tableur;

public class CelluleVide extends Cellule {
  @Override
  public int évaluer() {
    return 0;
  }

  @Override
  public String toString() {
    return ".";
  }
}
```

4. Complétez le code de la classe `Tableur` et écrivez une courte application affichant un tableur de taille 5 sur 5.

**Solution:** Les trois méthodes `getNbLignes`, `getNbColonnes` et `coordCorrecte` sont optionnelles et peuvent être omises dans un premier temps. Toutefois, elles facilitent l'expression de la précondition de la méthode `set` et `coordCorrecte` sera aussi employées dans les questions suivantes pour la vérification de préconditions d'une autre méthode.

```
// Tableur.java
package tableur;

public class Tableur {
  private Cellule tab[][];
```

```
  public Tableur(int nbLignes, int nbColonnes) {
    assert(nbLignes > 0 && nbColonnes > 0);
    tab = new Cellule[nbLignes][nbColonnes];
    for (int x = 0; x < nbLignes; ++x)
      for (int y = 0; y < nbColonnes; ++y)
        tab[x][y] = new CelluleVide();
  }

  public int getNbLignes() {
    return tab.length;
  }

  public int getNbColonnes() {
    return tab[0].length;
  }

  public boolean coordCorrecte(Coord c) {
    return 0 <= c.getColonne() &&
           c.getColonne() < getNbColonnes() &&
           0 <= c.getLigne() &&
           c.getLigne() < getNbLignes();
  }

  public void set(Coord c, Cellule cell) {
    assert(coordCorrecte(c));
    tab[c.getLigne()][c.getColonne()] = cell;
  }

  public String toString() {
    String s = "";
    for (Cellule[] t : tab) {
      for (Cellule c : t)
        s += c + "␣";
      s += "\n";
    }
    return s;
  }
}

// AppliTableur.java
import tableur.Tableur;

public class AppliTableur {
  public static void main(String[] args) {
    final int MAX = 5;
    Tableur t = new Tableur(MAX, MAX);

    System.out.println(t);
  }
}
```

5. Définissez et codez la sous-classe représentant les cellules constantes et complétez votre application pour tester ce nouveau type de cellule.

**Solution:**

```
// CelluleConstante.java
package tableur;

public class CelluleConstante extends Cellule {
  private int valeur;

  public CelluleConstante(int v) {
    valeur = v;
  }

  @Override
  public int évaluer() {
    return valeur;
  }
}

// AppliTableur.java
import tableur.Tableur;
import tableur.CelluleConstante;
import tableur.Coord;

public class AppliTableur {
  public static void main(String[] args) {
    final int MAX = 5;
    Tableur t = new Tableur(MAX, MAX);
    t.set(new Coord(0, 0), new CelluleConstante(1));

    System.out.println(t);
  }
}
```

6. Définissez et codez la sous-classe représentant les cellules sommant deux autres cellules. Ceci peut vous conduire à introduire de nouvelles méthodes dans la classe `Tableur`. Complétez votre application pour tester ce nouveau type de cellule.

**Solution:** Une telle cellule a besoin de pouvoir accéder aux cellules d'un tableur. Un accesseur doit donc être ajouté à la classe `Tableur`. Vous noterez que toutes les préconditions ne sont pas vérifiées. En particulier, des références circulaires peuvent être introduites par les sommes. La détection de ce type d'erreur dépasse le cadre de cette séance mais peuvent donner lieu à un très bon exercice d'algorithmique...

```
// Tableur.java
package tableur;

public class Tableur {
  ...
  public Cellule get(Coord c) {
    assert(coordCorrecte(c));
    return tab[c.getLigne()][c.getColonne()];
  }
  ...
}

// CelluleAddition.java
package tableur;
```

```
public class CelluleAddition extends Cellule {
  private Tableur t;
  private Coord c1, c2;

  public CelluleAddition(Tableur t, Coord c1, Coord c2) {
    assert(t.coordCorrecte(c1) && t.coordCorrecte(c2));
    this.t = t;
    this.c1 = c1;
    this.c2 = c2;
  }

  @Override
  public int évaluer() {
    return t.get(c1).évaluer() + t.get(c2).évaluer();
  }
}


// AppliTableur.java
import tableur.Tableur;
import tableur.CelluleConstante;
import tableur.CelluleAddition;
import tableur.Coord;

public class AppliTableur {
  // Affiche le triangle de Pascal
  public static void main(String[] args) {
    final int MAX = 10;
    Tableur t = new Tableur(MAX, MAX);

    t.set(new Coord(0, 0), new CelluleConstante(1));
    for (int i = 1; i < MAX; ++i) {
      t.set(new Coord(i, 0), new CelluleConstante(1));
      for (int j = 1; j <= i - 1; ++j)
        t.set(new Coord(i, j), new CelluleAddition(t,
            new Coord(i - 1, j - 1), new Coord(i - 1, j)));
      t.set(new Coord(i, i), new CelluleConstante(1));
    }
    System.out.println(t);
  }
}
```

# Annexe

```java
// Coord.java
package tableur;

public class Coord {
  private int numL, numC;

  public Coord(int ligne, int colonne) {
    super();
    this.numL = ligne;
    this.numC = colonne;
  }

  public int getLigne() {
    return numL;
  }

  public int getColonne() {
    return numC;
  }
}
```